

サーバプッシュ技術サーベイ及びHTMLブラウザでの双方向通信について

HTMLブラウザを組込み機器のインタラクティブなUIとして活用する基盤技術
2008/1/9

HTMLブラウザを組込み機器のフロントエンドとして使用するために必要な技術として、「HTMLブラウザにおける双方向通信」について述べる。

まず、その導入としてサーバプッシュ技術についてまとめる。なぜなら、双方向通信を導入する要件は、情報のサーバプッシュであり、その技術そのものが双方向通信を実現するにあたってのキーと見なせるからである。

サーバプッシュについて説明した後、HTMLブラウザにおける双方向通信について述べる。

サーバプッシュ技術サーベイ

あらためて現状のウェブ技術関連におけるサーバプッシュ技術について概観する。

ウェブにおけるサーバプッシュ技術がいわれて久しい。しかし、ウェブのサーバプッシュ技術は普及している状態とはいえない。メッセージングやIPフォンなどのアプリケーションではサーバプッシュ型の技術が使われているがHTTPベースのアプリケーションでは広く使われていない。

色々な理由が考えられるが、プロトコルの問題、セキュリティの問題のふたつが大きな理由と考えられる。

HTTPプロトコルは、リクエスト・レスポンスプロトコルである。「情報の取得には必ずクライアント側からのリクエストが必要」である。GET/POSTなどクライアント側のHTMLブラウザからのHTTP要求をHTTPサーバが受け取り、それに対してHTTPサーバがHTTP応答を返すという繰り返しからなっている。

つまり、サーバ側から一方的にブラウザ側に情報を通知できない。サーバ側からイベント的になんのクライアント側からのリクエストもなく一方的に接続を確立したり、データを送りつけてくることはできない。

HTTPを使ったサーバプッシュの難しさはこのプロトコルの特性から生じている。HTTPで取り扱うには、サーバプッシュ技術はそもそも向かない問題といえる。また、ソフトウェア基盤の観点からしても、HTTPをベースにしたウェブシステムを支える様々なソフトウェア(サーバ、プロキシ、ブラウザなど)が当プロトコルの特性に最適化されて作られてきているため、サーバプッシュをその中で実現しようとしても難しい。

また、この特性とNATのような技術の組み合わせによって、ブラウザ側はグローバルIPを持たないクライアントに徹する形のシステムが大多数となっている。様々なセキュリティ要件によって、多くの市販ルータではサーバとするには別の設定が必要な状態となっている。

しかしながら、現状、膨大なインストールベースが存在している。その既存のシステムの中でいかに効率よくサーバプッシュを実現するかが課題である。

HTTPブラウザとHTTPサーバしかなく双方間でHTTPによる一方的な通信のみが可能
なその既存のシステムの中での実現を検討する。

課題を解決する手段として以下の5つの方法が挙げられる。よく知られている方法もあ
れば、あまり知られていない方法もある。

1. Polling: ポーリング方式
2. Long poll: 長時間ポーリング方式
3. Chunked data: チャンク方式
4. Embedded Plugin: 埋め込みプラグイン方式
5. Server-sent DOM events: イベントソース方式

以下ではこれらの5方式について説明し、得失を概観する。

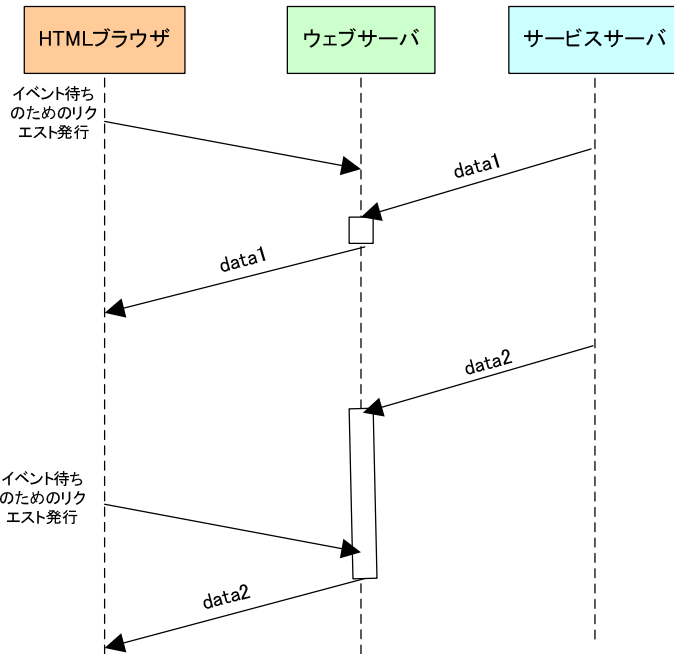
説明では、HTMLブラウザ・ウェブサーバ・サービスサーバの3つのオブジェクト間のシー
ケンスで説明を加えている。HTMLブラウザ・ウェブサーバについては説明をしない。サ
ービスサーバとはここではなんらかのネットワークアプリケーションサービスを実現してい
るサーバとしている。特に、サービスサーバは、要求-応答の組によるサービス提供だけ
ではなく、HTMLブラウザ・ウェブサーバからのトリガなしに、サービスサーバからイニシエ
ートされるようなイベント的なデータ送信を行うものとする。

1. Polling: ポーリング方式

インターバルごとに情報をブラウザからサーバに 要求する。

1.a Meta tag: メタタグにrefreshインターバルを記述しておき、インターバルごとにペー
ジ更新。

1.b Loop and timeout: ページ内のJavascriptで無限ループ。
タイムアウト時にjavascriptタグの評価を行い、評価結果に応じてHTTPリクエストを発
行する。



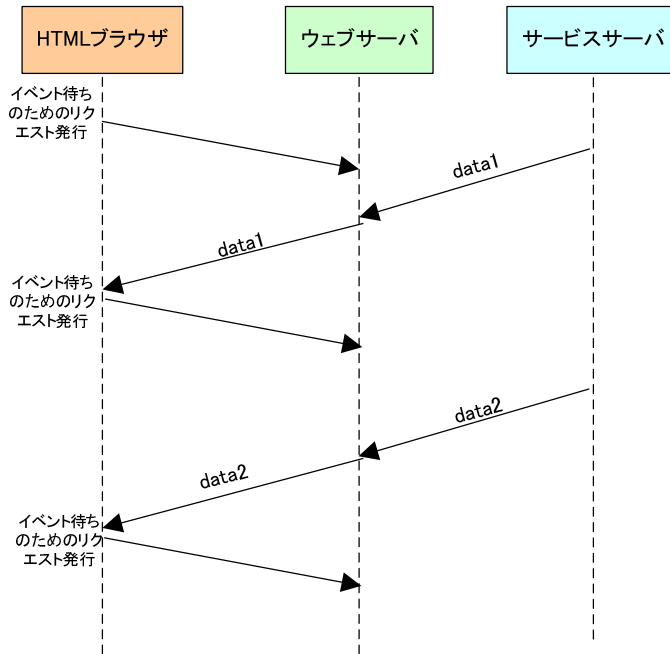
インターバルごとにブラウザからサーバにHTTP要求を発行しその度に応答する仕組みのため、以下のような強み・弱みがある。

- 実装が単純であり、またサーバ側のアプリの変更の必要性が低い
- 要求の間隔が長いとタイムラグが長くなりリアルタイム性が損なわれる
インターバルがユーザにとってはレイテンシに感じられる。(上記、シーケンス図中、白い四角の部分)がレイテンシである。
- 間隔を短くするとサーバやネットワークの負荷が大きくなる

クライアントの数が増えると接続回数が急激に増え、サーバ側での対応が不可能になるなどスケーラビリティに問題がある。サーバの限界が近くなった場合にはクライアントからの要求のインターバルを長くすることによって対応可能数を増やすことができる。しかしその場合はイベントへの反応のレイテンシが増すのはいうまでもない。したがって、インターバルを十分に確保できるコンテンツに向く。

2. Long poll: 長時間ポーリング方式

通知用の接続を確立しておき、情報通知のイベントが発生したとき、その通知用の接続を用いて返却する。通知時にリクエストがいったん終息、再度リクエストを発行する。



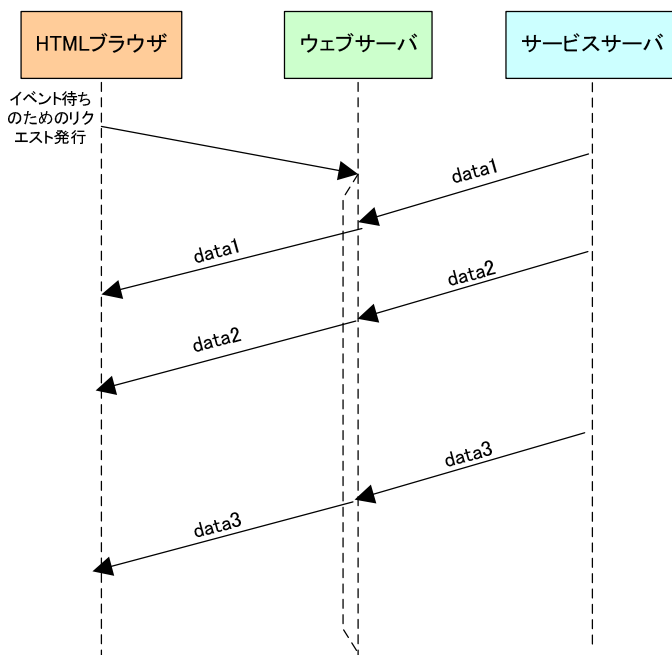
サーバ側でのイベントごとあるいは接続のタイムアウトごとに新しいコネクションを張りなおすことになる。

- サーバ側でのアーキテクチャの変更が大きい。特にウェブサーバでの対応が必要。
- イベントが輻輳してきた場合、新しいコネクションを張ってもすぐにイベント通知に使用され、実質的にLong pollにならず、ネットワークリソースを消費する。
 - したがって、単位時間あたりのイベント数に対してのスケールビリティが弱い。一般にクライアントの数が増えるとイベント数も増える傾向にある。特に、クライアントの数が増えるほどイベントの数が急速に増加するものの場合、急激にレスポンス性が低下することが懸念される。したがって当方式の場合、Polling方式のようにインターバルを調整できないため、クライアント数の上限が低いと考えられる。

3. Chunked data: チャンク方式

HTTP/1.0のHTTP接続ではHTMLブラウザがリクエストをサーバに送ると、サーバはレスポンスを1つ返す。ひとつのコネクションでひとつの要求とひとつの応答を得ることができる。一方、HTTP/1.1では、様々な方式でひとつのコネクションで複数の要求・複数の応答が可能になっている。ここでは特にchunkedデータに着目するが、Transfer-Encoding: chunkedヘッダがつけられたレスポンスにおいては、レスポンスを複数個返すことができる。Chunked dataチャンク方式とは、このchunked形式によって、複数個レスポンスを返す方式を使い、長時間接続を維持しながら、レスポンスをイベントごとに返すという形にしたもの。

Transfer-Encoding: chunkedおよびContent-type: multipart/x-mixed-replaceをヘッダにつけ、データをchunked形式のかたまりごとに返すことになる。



ウェブサーバは、サービスサーバから受け取ったデータをひとつのコネクションで複数の塊としてチャンクあるいはマルチパートに含めて応答する。テキストでもxmlエンベロープでもよい。

ひとつの応答後も後続の応答のためにコネクションはそのまま維持される。通知用のコネクションを確立しておき、情報通知のイベントが発生したとき、その通知用のコネクションを用いて返却する。

ここまではLong Poll式と同様。ただし、HTTP応答としてはそのまま継続する。

リクエストは再発行しない。

コネクション確立のためにXmlHttpRequestを使用するが、XmlHttpRequestはドメインを越えて(同一ホスト・異なるポートへの接続もできない)もは接続できないため、通知用のコネクションの維持にあたっては、ウェブサーバでの処理が必要となる。

具体的な応答内容は以下のとおりである。

サーバ側からの応答の例

```

HTTP/1.1 200 OK
Transfer-Encoding: chunked
Content-Type: multipart/x-mixed-replace;boundary="rn9012"

7e
--rn9012
Content-type: application/xml

<?xml version=' 1.0' ?>
<content>Multipart: First Part of Request 10</content>
--rn9012
  
```

78

Content-type: application/xml

<?xml version='1.0' ?>

<content>Multipart: Second Part of Request 10</content>

--rn9012--

0

サーバ側は、HTTP応答時に、Content-Typeとして、chunked、もしくは、application/xmlを用いる。データのひとかたまりをchunked形式またはmultipart/x-mixed-replaceの形式で返却する。

HTTPブラウザ側では、XMLHttpRequest#responseTextとして受け取る。HTTPブラウザ側のXMLHttpRequestの実装に依存しており、現時点で、PC用のブラウザでは、IEは対応不可、Mozilla,Operaのみが対応可となっている。

responseTextは、一要求に対するすべての応答内容を保持しているため、新しい応答部分は、直前の応答との差分部分として認識することがアプリ側で可能である。また、responseTextは読み込みのみのプロパティであるため、responseTextが必要以上に大きくなってしまった場合には、いったん接続を切って再接続を行いresponseTextをアプリ側でリセットする必要がある。

4. Embedded Plugin: 埋め込みプラグイン方式

objectタグもしくはembedタグを使用しネットワークプロキシ機能をもった埋め込みプラグインを用いる方式

ネットワーク通信のバックエンドとして埋め込みプラグインを用いる。プラグインをソケット通信のプロキシとして用いブラウザを直接インタラクティブなサービスを実施するサーバ(サービスサーバ)に接続させる。ページに不可視のプラグインオブジェクトを埋め込み、プラグインはソケットでサービスサーバのサービスのポートへ接続し、HTMLブラウザではない直接接続のクライアント同様に振る舞う。ブラウザ側からはJavascriptを通して当該プラグインを制御する。特に、HTTPの背万ティクスに応じた処理が必要ではないので、シーケンスは省略する。

埋め込みプラグインとしてPCの場合Flashを前提としてよいほどインストールベースが広がっておりFlashとブラウザの親和性が高くなっている。Flashを使用する場合、APIとしてはXMLSocketもしくはsocketが利用できる。Flashは出来る限りネットワークコネクションを確立する役割に留め、プレゼンテーション層・ロジック層についてDHTMLにて対応することにより、Flashの利用をユーザに悟られないようにすることが可能。

- FlashPlayerのセキュリティポリシーによってはユーザにダイアログが提示される。(ネットワークに接続してよいか? など)
- FlashPlayerがないブラウザもありえる。

5. Server-sent DOM events: イベントソース方式

当方式は、HTML 5として提案されている規格であり、Opera 9.xでのみ利用できると言われている。以下の動作はOpera 9.25 build8827で確認した。

Server sent DOM eventは、long poll型のアクセスを可能にする仕組みである。ブラウザはevent-sourceタグのsrc属性に書かれたリソースにアクセスする。リソースへの接続が失われた場合、ブラウザは短時間(5秒以下を目安)に再接続を実施する。

- Content-Typeとして、application/x-dom-event-stream を指定する。
- コンテンツは、連続したイベントを含むことができ、以下の形式をしている。(厳密には仕様書を参照されたい)

行の繰り返し

行は、名前: 値¥n の形式、または ¥nである。

イベントは連続する複数の行からなり、改行(¥n)のみの行までで一区切りである。一区切りのイベント中、名前が一致する場合は、改行を含めて値が結合される。

- サーバ側からDOM Eventの送付が可能。
名前の部分が、Event、Class、Namespace、Bubbles、Cancelable、Targetの場合、特定の振る舞いを行う。
- たとえば、Targetが指定された場合、HTMLファイル内の特定のタグをIDによって示すことができる。

サーバから送られてくるイベントの例

```
Event: stock change
data: FUJITSU
data: -2
data: 10
```

スクリプティングの例

```
<event-source src="http://127.0.0.1/es.cgi" id="stock">
<script type="text/javascript">
document.getElementById('stock').addEventListener('stock
change',
function () {
var data = event.data.split('¥n');
alert(data.join(' '));
}, false);
</script>
```

サーバ側プログラミングの例

Webrickによるサンプルサーバ
es.cgiにアクセスするとes.rbが起動される。

```
srv = WEBrick::HTTPServer.new({:DocumentRoot => '/tmp/',
                               :BindAddress => '127.0.0.1',
                               :Port => 8181});
srv.mount('/es.cgi', WEBrick::HTTPServlet::CGIHandler, 'es.rb');
srv.start;
```

CGIの例(es.rb)

```
#!/usr/bin/ruby
sleep 3
print "Content-Type: application/x-dom-event-stream\r\n\r\n"
print "Event: stock change\r\n\r\n"
print "data: FUJITSU\r\n\r\n"
print "data: -2\r\n\r\n"
print "data: 10\r\n\r\n"
exit 0
```

HTML5のConnectionタグ

HTML 5においては、Connectionタグによるネットワーク接続がサポートされる見込みである。セキュリティ上の課題が存在しているため、ローカルエリアネットワークに閉じるなど限定的なサポートにとどまると考えられるが、特殊なプロキシサーバを用意することによって、常時接続型のDHTMLアプリケーション・サービスを多数登場させるだろう。

各方式の得失のまとめ

Embedded Plugin方式を除く方式は、DHTML技術をコアにしたサーバ側アプリケーションとHTMLブラウザによる対応とからなっている。従って、クライアントとしては、ブラウザ無依存にはなっていないものの、プラグインを使用せずHTMLブラウザのみに閉じた形で実現できる。

この場合、利点としては以下が挙げられる。

- ブラウザの中で完結するため、他のDHTMLを使用したものと統合を実施しやすいJavaアプレットよりも動作が軽い。
- CORBA/RMIと違いHTTP向けに設定されているファイアウォールを越えることができる。CORBA/RMIサーバが不要。
- プロトコル上のオーバーヘッドが小さい

欠点としては以下が挙げられる。

- 複数のブラウザ互換をとるのは難しい。
ブラウザ毎に非互換性がありそれらを吸収したJavascriptライブラリが多数開発されているものの、ブラウザコア部分での非互換があると そういったライブラリでは吸収できずコアの修整が必要となる。特にChunked data方式については大きなユーザベースを持っているブラウザ(IE:シェア90%以上)での非対応(readystate==3のときのresponseTextの値がnull)であるためにこれを前提とすることはPCの世界では難しい。event-sourceについてはサーバとのネゴシエーション・コネクション継続の仕様が不明であり安定して利用できることは現状考えにくい。HTML5におけるevent-sourceタグのサポートは現状Operaのみとなっており、これもブラウザ依存の現状である。
- スケーラビリティを確保できない。

プッシュ技術の実装によってはたかだか100個程度の クライアントの対応でサーバがサービスレベルを維持できなくなる。

- プログラミングが煩雑となる。
2本のコネクションを使って処理を行うことになるため、クライアント側・サーバ側ともプログラミングにあたってはコネクション維持のハンドリング、データの送受信時において注意が必要となる。
- ウェブサーバの設計の問題：
ウェブサーバは一般に長い寿命を持つコネクションに対応できるように設計されていない。
- プロキシバッファリング：
ある種のプロキシは通信をバッファリングしてしまうため イベント発生ごとのプッシュにならない。

	プラグイン不要	ブラウザ依存性	スケーラビリティ	プログラミングの容易さ
ポーリング方式	○	○依存性無	×	○
長時間ポーリング方式	○	○依存性無	△イベントごとに再接続が発生	×
チャンク方式	○	×(Mozillaのみ)	△	×
埋め込みプラグイン方式	×(プラグイン要)	○依存性無	○(HTTPサーバを使用せず独自サーバを利用)	×
イベントソース方式	○	×(Operaのみ)	△	○

現状のHTMLブラウザにおける双方向通信について

先の節において、サーバプッシュの方法を概観したが、これらのHTML4をサポートした現状のHTMLブラウザにおける双方向通信の観点で整理すると以下の2方式となる。

A. 上り下りの2接続を使ってプロキシサーバに接続し擬似的に双方向通信を構成する当該プロキシサーバを擬似コネクションプロキシサーバと呼ぶとする

- Polling, Long Poll, Chunked data, sever-sent DOM events 各方式がこれにあたる

これらの各方式をすべて考慮し議論するのは煩雑であるので、ここでは、Chunked data およびsever-sent DOM eventsによって実現可能な方法である、継続的に接続した上り下りの2接続を使う方式をAとする。

より詳細化すると、A方式とは、POSTにより情報を送信する上りと、あらかじめ設定してあるコネクションを通じてひとつのエンベロープにて一塊の値を返却する下りの2つのコネクションによってひとつの転送路をエミュレートする方式をいうとする。

長所を述べると繰り返しになるが、HTTPブラウザのみで実現可能であることに尽きる。一方、ネットワークリソースをそれなりに使用するので、そもそもプロキシがどれぐらいスケ

ールするかは実装とそのサービスの特性に依存する。また、コネクションがプロキシ経由となるためネットワークの本来のエンドポイントがサービスサーバ側から見えないため、大規模なネットワークでは、セキュリティ上好ましくない状態として、クライアントの接続が切られるサービスもありえるだろう。

B. 埋め込みプラグインをネットワークプロキシとして使用する。

- Embedded Plugin方式がこれにあたる
(およびHTML5 Connection方式がこれにあたる)

現状、双方向通信実現にあたっての問題のポイントは以下の2つである。

— ポータビリティ

— A方式の場合

ポイント:ブラウザの違い:

- chunked data方式の場合、チャンク受信ごとにonreadystatechangeが呼び出され、XMLHttpRequestのreadystate==3もしくは4の時のresponseTextの値が設定されるかどうか
- イベントソース方式の場合、ブラウザがevent-sourceが使用できるかどうか

• XMLHttpRequestのコネクションをアプリのコンテキストに合わせて維持し続ける必要がある。

• イベントソース方式のコネクション維持ロジックが不明である。

— B方式の場合: Flashを使う場合と使わない場合があると考えられる。

ポイント:FlashPlayerの搭載有無・FlashPlayerのバージョンが目的と合うかどうかなどが問題となろう。

ポイント:ブラウザごとに異なるAPI

- 独自に埋め込みプラグインを開発する場合、プラグインへのサイン・ブラウザごと埋め込みプラグインの開発(様々なブラウザのJavascriptとのインタラクションを開発しなければならない)

— スケーラビリティ

— A方式の場合、擬似コネクションプロキシサーバの性能・品質および負荷分散が課題となる。

— B方式の場合、サービスサーバのスケラビリティがそのまま制約条件になるのでここでは議論しない。

HTMLブラウザとサービスサーバ間の双方向通信をサポートしたアプリケーションモデル

以下の図は、HTMLブラウザとサービスサーバ間の双方向通信をベースとしたアプリケーションモデルを記述したものである。

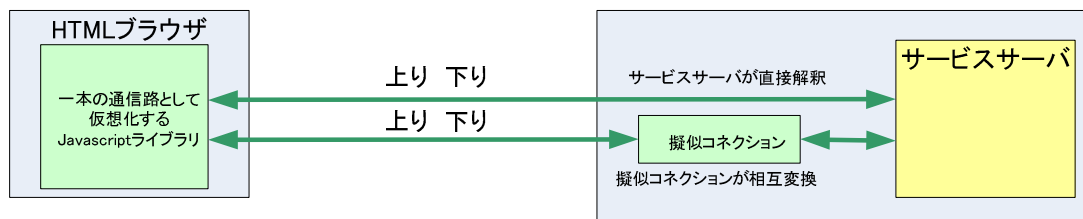
アプリケーションのビジネスロジックはサービスサーバ側で実装されている。

一方、プレゼンテーション層についてはHTMLブラウザにまかされている。

このふたつの大きな箱が両方とも組み込み機器に内蔵されていることに注意されたい。

このふたつを繋ぐプロトコルは当該アプリケーションが直接サポートすることも可能であるし、擬似コネクションプロキシサーバで変換することも可能である。

模式図



たとえば、IRCのように既存の十分に品質のとられたサーバ(サービスサーバ)が存在している場合、擬似コネクションプロキシサーバでデータ変換(POSTデータからIRCの命令への変換およびその逆)を行うのが適切だろう。

レスポンスのスケールビリティが必要で、サービスサーバのクラスタリングが効くような場合でかつサービスサーバが自前の場合は変換処理はサービスサーバ側で実施という選択もありえる。

様々なパターンが考えられるが、効率的に開発を進めようとするならば、HTMLブラウザとサービスサーバ間で互いにRPCをしながら処理を進める基盤を用意する必要がある。

HTMLブラウザの双方向通信の具体的な方式提案

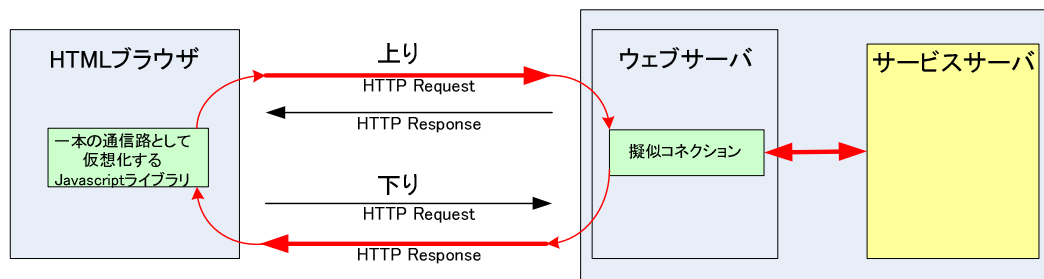
以下、HTMLブラウザの双方向通信についてより具体的な3方式を挙げる。

組込み機器内に内蔵のウェブサーバ(HTTPサーバ)およびサービスサーバを用意するとする。ブラウザとウェブサーバ、ブラウザとサービスサーバ間はsocketを用いる。

A方式：擬似コネクションプロキシサーバ

双方向コネクションを実現する通信ミドルウェアとして、擬似コネクションプロキシサーバ機能を持つ双方向コネクションモジュールをウェブサーバ内に置く。

当モジュールについては、ブラウザとまったく独立して開発できるのが強み。ただし、ネットワークのエンドポイントがサービスサーバ側から見えないというのは組込みでも変わらない。双方向コネクションモジュールが複数のコネクションを保持することになるが、そのネットワークハンドリング方式が性能とスケールビリティに大きく影響する。



- ・当方式の場合、上りをHTTP Request Pipingによって継続的に送り出す方法が考えられるが、現状Javascriptからそれを制御できる方法が用意されていない。
- ・擬似コネクションを実現するウェブサーバまたはウェブサーバ側モジュールについてはここでは詳細については説明しないが、いわゆるC10K問題への対処が必要なものとなる。従来のウェブサーバは、長時間のコネクションを保持する仕組みになっていないため、長時間のコネクションを保持が一般には重い処理となっている。従って、双方向コネクションモジュールが主たる機能となる場合は、ウェブサーバ基盤そのものの改変が必要

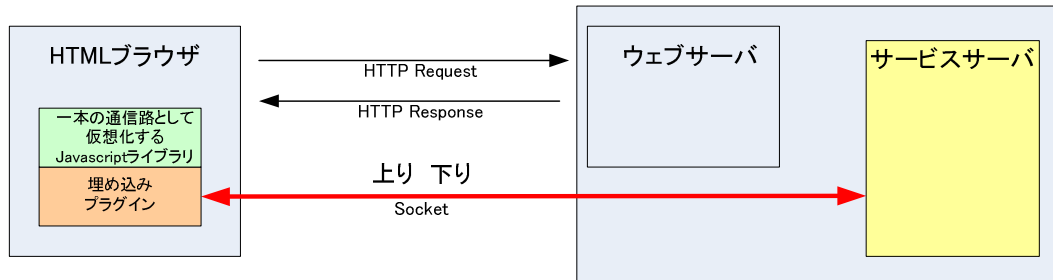
である。

B方式：埋め込みプラグイン

ブラウザに密着しているのが強み。

ネットワーク上のエンドポイントがサーバ側からそのまま接続元に見える。

複数のネットワーク接続を保持するための埋め込みプラグイン開発が必要。



C方式：HTML5のConnectionを先取り

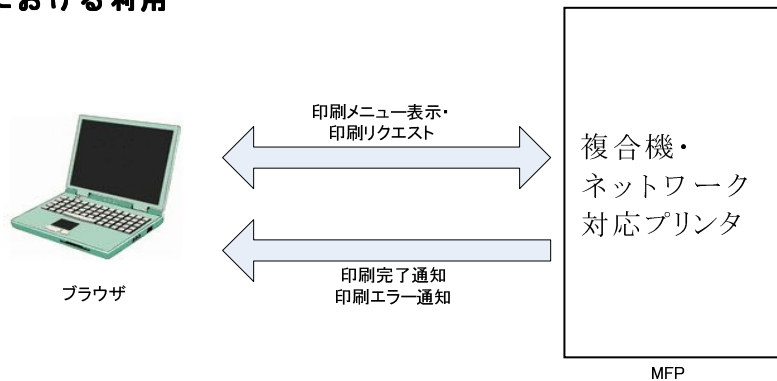
HTMLブラウザのコアをカスタマイズし、HTML5のConnectionを先取りして実装。

セキュリティモデルが不明なのでこれ以上の議論はここではしない。

想定される利用シーン

以下では、HTMLブラウザによる双方向通信について、想定される利用シーンを挙げる。

MFPにおける利用



昨今、オフィスにはネットワーク対応のプリンタ(MFP)が置かれるようになってきた。現状は、WindowsのネットワークファイルアクセスやWindowsのネットワークプリンタとしてプリントの指示が可能という状態である。また、MFPがファイルサーバの役割を果たすことも多くなりつつある。MFPはスキャナ機能を持つため、スキャンした電子データをそのまま格納しておくことが可能である。こういったデータをいちいちローカルパソコン側に取り出すことなくMFP上で印刷できるというのは使い勝手がよい。

ブラウザによる双方向アクセスが可能となると以下のようなことが可能となる。

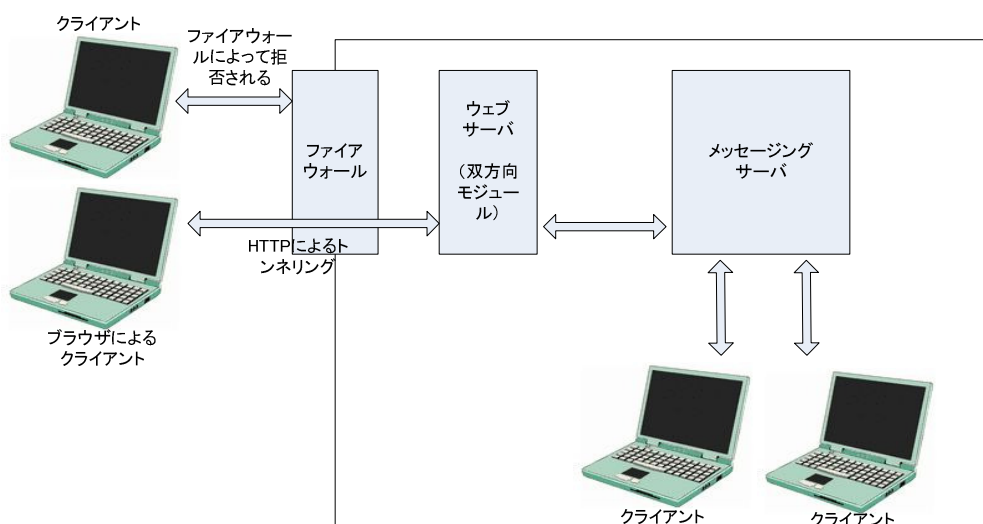
PCにインストールされたブラウザで、(LAN外の)MFPにアクセス、MFP内に格納されたファイル一覧を取得、リクエスト発行を実施する。その後、印刷完了やエラー通知はイベントとしてプリンタ側から送信される。

		UI	ファイル管理	印刷プロトコル
従来	ローカルファイル	Windows標準	Windows SMB	Windows 印刷プロトコル IPP
提案	ローカルファイル	Windows標準	Windows SMB	Windows 印刷プロトコル IPP
	リモートファイル	DHTML(ブラウザ)	WebDAVや独自ウェブアプリ	独自ウェブアプリからの指示

メッセージング

リアルタイムメッセージングサーバ(IRCサーバなど)がファイアウォール内に置かれている場合、ファイアウォールの設定によって当メッセージングプロトコルを通さない設定をされている場合がある。

双方向接続をHTTPを通して使用し、HTTPを通すファイアウォールを越えて、ファイアウォール内に置かれているメッセージングサーバへアクセス可能とする。



この場合、ファイアウォールが長時間接続を保持するかどうかはその実装や設定に依存していると考えら得る。様々なファイアウォール製品における実装状況・実際の運用での設定状況について不明である。

双方向通信をサポートするブラウザ側基盤およびサーバ側で双方向接続を実現するウェブサーバと双方向接続モジュール-これらの基盤技術を使用して、組み込み機器UIにおいて、HTMLブラウザを利用したインタラクティブなアプリケーションを実現可能である。

まとめ

HTMLブラウザにおける双方向通信のサポートは、クライアント-サーバ間のインタラクションをより緊密にした、よりサービスに寄ったよりユーザビリティの高いアプリケーションの基盤となるだろう。

参照サイト

http://zzz.zggg.com/j/server_push_today_marginal_ajax.html

<http://www.pushlets.com/>

<http://www.hixie.ch/specs/html/server-sent-events/server-sent-events>

<http://www.whatwg.org/specs/web-apps/current-work/>

<http://www.kegel.com/c10k.html>

当資料は、HTMLブラウザをアプリケーションのUIとして使用するために必要な技術についての概要説明です。

当資料の内容について正確性については相当の配慮を行っていますが保証はいたしません。

当資料に含まれる商標・登録商標は各社の商標・登録商標です。